DOE Office of Science Facilities

ALCF, NERSC, OLCF

HPC Systems at the Office of Science

ALCF, NERSC and OLCF have begun transitioning users to energy-efficient architectures in preparation for Exascale

OLCF Titan
18,000+ CPU+GPU Nodes

ALCF Theta
2500+ Xeon-Phi Nodes

NERSC Cori
9500+ Xeon-Phi Nodes

http://performanceportability.org/facilities/overview/

Challenges and Opportunities

Users of these ALCF, NERSC, OLCF are scientists who first and foremost care about scientific results. They often have accounts and allocations at multiple Office of Science Facilities. They desire:

1. Portability - The want to be able to maintain a single branch of code that works across the different systems they have access to.

2. Longevity - They don’t want to have to rewrite their codes for each new system they can access to every couple of years.

What Recommendations do We Have For Them?

Challenges and Opportunities

Performance comes from “similar” architectural features

- Increase parallelism (Cores, Threads, Warps/SMs/Blocks)
- Vectorization (AVX512 8 Wide Vector Units, 32 Wide Warps)
- Small Amount High-bandwidth Coupled with Large Amounts of Traditional DDR

Portable Implementation Challenges:

1. How to express parallelism in a portable way across both the KNL processor cores and vector-lanes and across the 896 SIMT threads that the K20x CUDA cores support.
2. How to express data movement and data locality across the memory hierarchy containing both host and device memory in portable way.

Agree? Missing something?

DOE SC Facility Definition

An application is performance portable if it achieves a consistent ratio of the actual time to solution to either the best-known or the theoretical best time to solution on each platform with minimal platform specific code required.

Agree?

Measuring Performance Portability

Measuring **Performance** Portability:

Bad Ways

1. Compare time-to-solution on one system vs another.
2. Compare ratio of actual app performance to peak system performance

Good Ways

3. Compare time-to-solution on each system against a well-known optimal implementation
4. Compare performance on each system against a relevant roofline-model ceiling on each system (We've included instructions for KNL and GPU)

Agree?

Selecting a Performance Portability Strategy (And Limits of Each)

Threads and Vectors (SMT, SIMT, SIMD).

1. SIMT ≅ SMT : What you tend to get when taking a GPU code and attempting a first pass portable version. This leaves SIMD on the GPU un-expressed. Leads to concept of coalescing.

2. SIMT ≅ SIMD : What you tend to get by default with OpenMP (!$OMP SIMD). Limits what you can vectorize on GPU to code with which the CPU can vectorize.

3. Use nested parallelism to map GPU SMs/Warps to CPU Cores/Threads and threads within Warps to Vector lanes. Still lose flexibility on the GPU.

Example: (QCD Dslash)

Implementation 1:

Use Kokkos to define simple parallel_for over lattice sites. Map SIMT Threads to SMT threads/cores on CPU.

Result: Great performance on GPU, Bad performance on KNL (compiler can't vectorize)

Implementation 2:

Use Kokkos to both parallelize over lattice sites and new level of inner-parallelism (multiple right hand sides) using threadVectorRange

Result: Add a little non-portable code to help compile explicitly vectorize C++/Kokkos complex data-type.

High Level Findings

Leading Performance Portability Approaches

Approach	Benefits	Challenges
Libraries	Highly portable, not dependent on compiler implementations.	Many GPU libraries (e.g. CUFFT) are C only (requiring explicit interafces to use in FORTRAN) and don't have common interfaces. May lock-in data layout. In many cases libraries don't exist for problem.
OpenMP 4.5	Standardized. Support for C, C++, FORTRAN and others. Simple to get started.	Limited expressibility (particularly on GPUs). Reliant on quality of compiler implementation - which are generally immature on both GPU and CPU systems.
OpenACC	Standardized. Support for C, C++, FORTRAN.	Limited support in compilers, especially free compilers (e.g. GNU).
Kokkos	Allows significant expressibility (particularly on GPUs.)	Only supports C++. Vector parallelism often left-out on CPUs.
DSLs	Highest expressibility for appropriate problems	Limited to only a small number of communities. Needs to be maintained and supported for new architectures.

Agree?

Recommendations

Available options for performance portability are still somewhat immature and evolving quickly.

- Profile your application using tools at:
- If you can use a well-supported Library or DSL, use it
- If your application doesn’t use C++, try OpenMP 4.5 and OpenACC (Later lacks a lot of testing on CPUs/KNL)
- If your application does use C++, consider Kokkos or Raja / OpenMP 4.5 (incremental) and which best fits your application
- Reach out to your facility to report bugs/deficiencies that we can share with the framework developers and standard bodies

Agree?

Extras

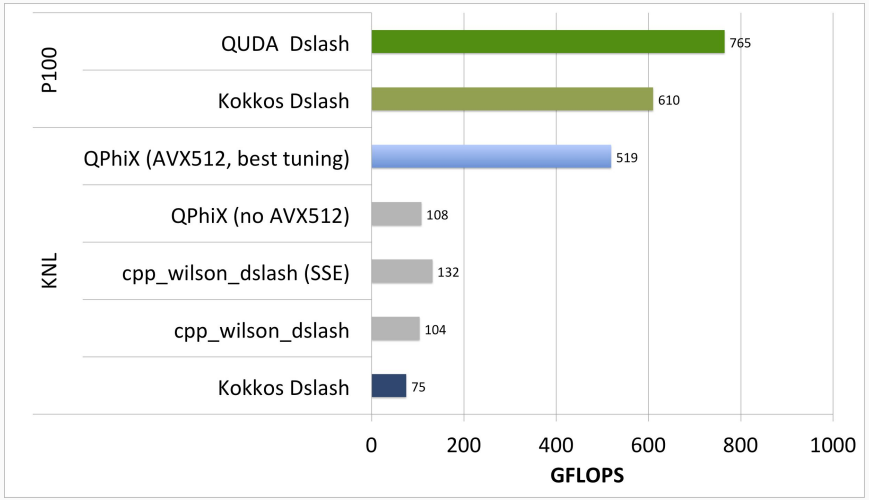
Feedback:

We are looking for feedback on definition, measurements, strategy, conclusions, recommendations

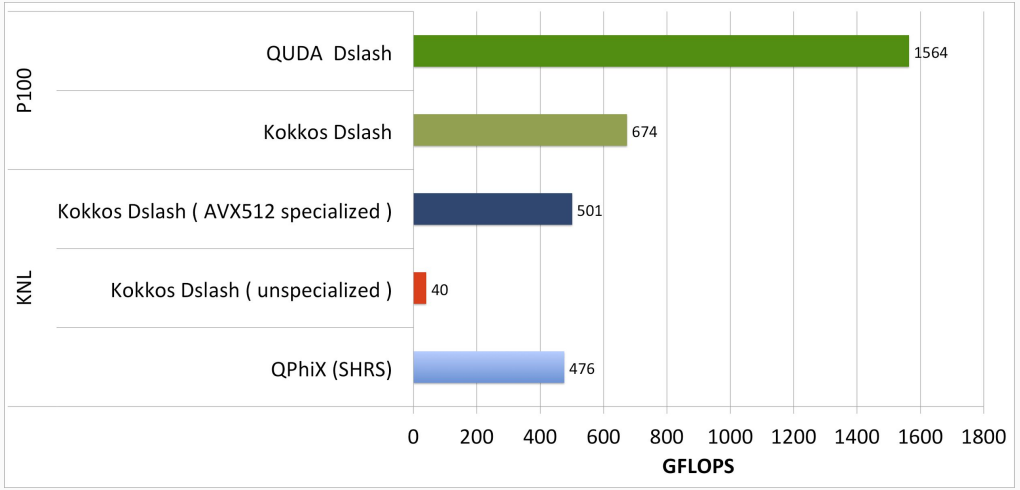
http://performanceportability.org

Example: (QCD Dslash)

Single Right Hand Side



Multiple Right Hand Side



http://performanceportability.org/case_studies/qcd/results_summary/

Defining Performance Portability

Definitions from Last Years Meeting:

- "For the purposes of this meeting, it is the ability to run an application with acceptable performance across KNL and GPU-based systems with a single version of source code." (Neely)
- "An application is performance portable if it achieves a consistent level of performance (e.g. defined by execution time or other figure of merit (not percentage of peak flops across platforms)) relative to the best known implementation on each platform." (Pennycook, Intel)
- "Hard portability = no code changes and no tuning. Software portability = simple code mods with no algorithmic changes. Non-portable = algorithmic changes" (Pope, Morozov)
- (Performance portability means) the same source code will run productively on a variety of different architectures" (Larkin)
- "Code is performance portable when the application team says its performance portable!" (Richards)

Measuring Performance Portability

Measuring Portability:

Percentage of code lines that are shared between architectures vs code lines that architecture specific.

Hard to measure if total code lines have gone up, but about as good as you can do.

Measuring Performance Portability

Roofline Data and Instructions

System	DRAM	L2	L1	GEMM
Titan (Kepler)	161 GB/s	559 GB/s	-	1226 GF/s
Summit Dev (4 Pascal)	1930 GB/s	6507 GB/s	-	17095 GF/s
Cori (KNL)	413 GB/s	1965 GB/s	6443 GB/s	2450 GF/s

http://performanceportability.org/perfport/measurements/

Collecting Roofline Data on KNL:

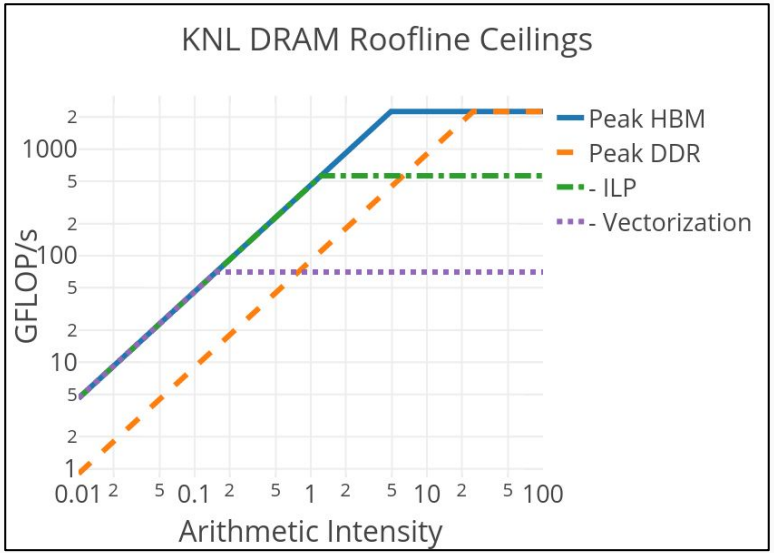
http://performanceportability.org/perfport/measurements/knl/

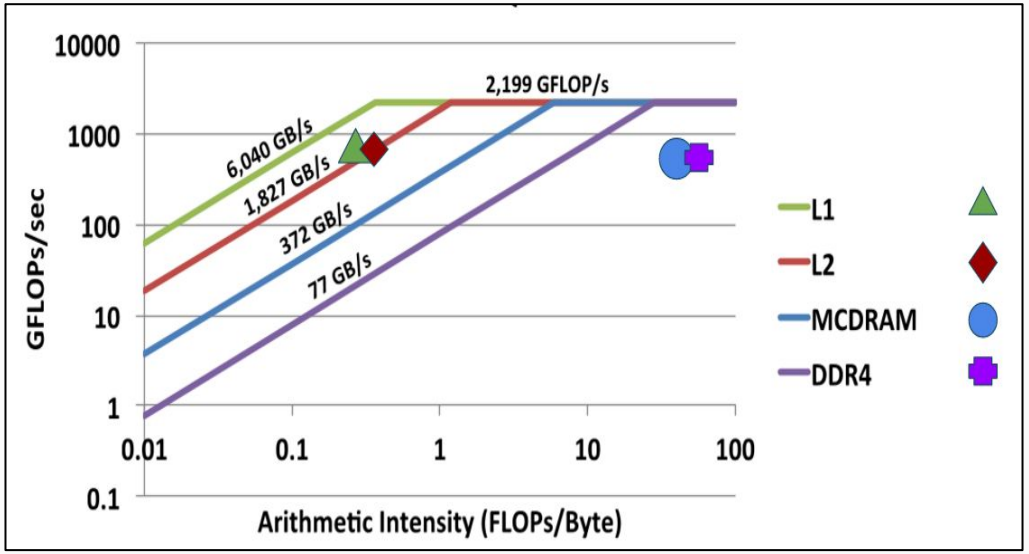
Collecting Roofline Data on GPUs:

http://performanceportability.org/perfport/measurements/gpu/

Measuring Performance Portability

Roofline Model (Find Most Relevant Ceiling)





One can extend the roofline to collect multiple relevant ceilings and AI values:

Different Levels of Cache/Memory, Different memory Access Patterns, Network Bandwidth/Traffic